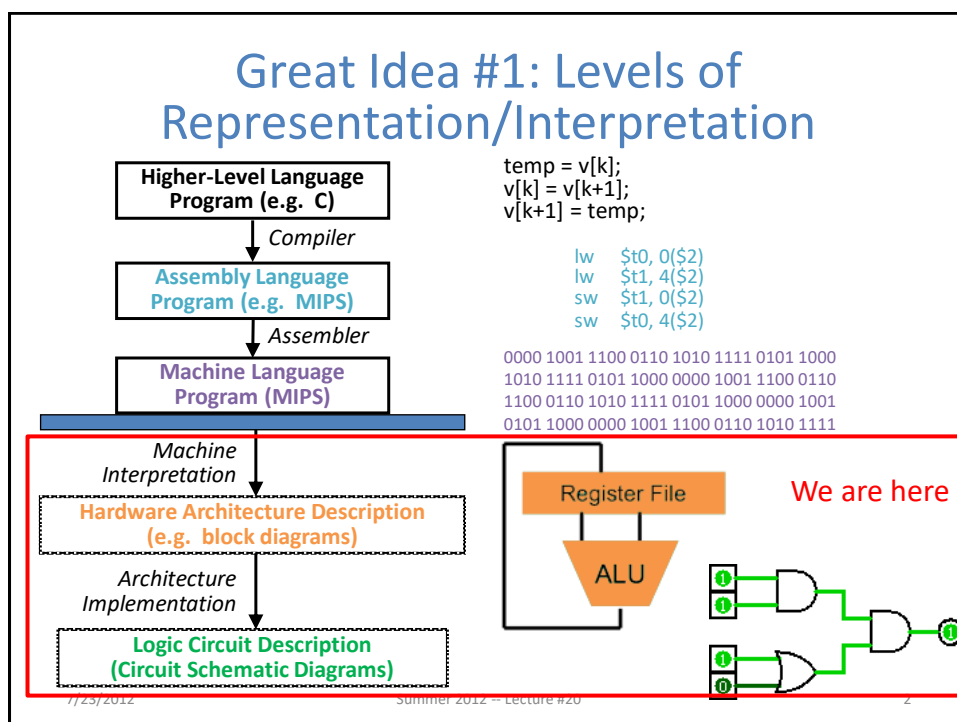


计算机组成

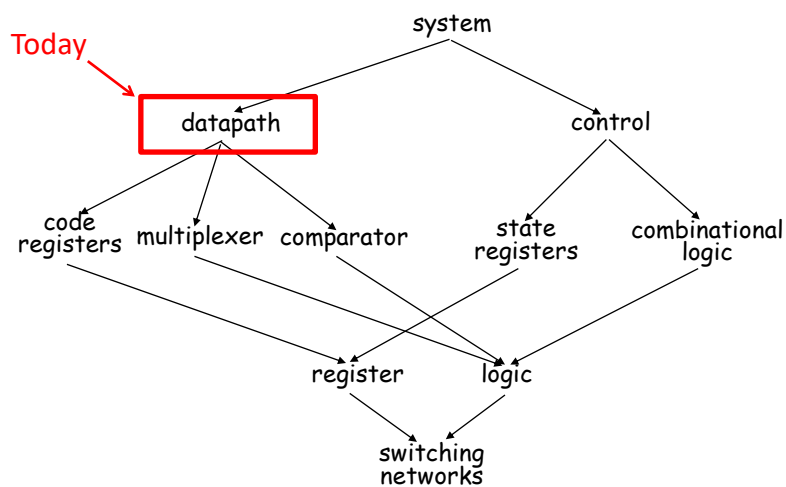
MIPS单周期数据通路

高小鹏

北京航空航天大学计算机学院



Hardware Design Hierarchy



7/23/2012

Summer 2012 -- Lecture #20

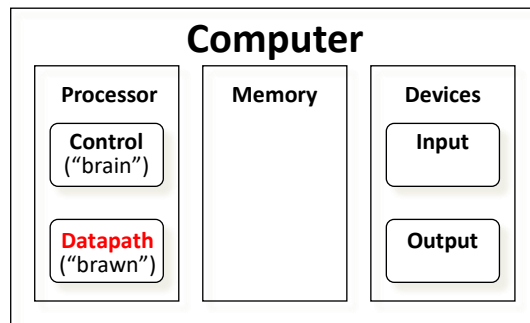
3

目录

- ❑ 概述
- ❑ 单周期CPU设计模型
- ❑ 数据通路基础部件建模
- ❑ 组装数据通路
- ❑ 控制介绍

Five Components of a Computer

- Components a computer needs to work
 - Control
 - Datapath
 - Memory
 - Input
 - Output



7/23/2012

Summer 2012 -- Lecture #20

5

The Processor

- **Processor (CPU):** Implements the instructions of the Instruction Set Architecture (ISA)
 - **Datapath:** part of the processor that contains the hardware necessary to perform operations required by the processor ("the brawn")
 - **Control:** part of the processor (also in hardware) which tells the datapath what needs to be done ("the brain")

7/23/2012

Summer 2012 -- Lecture #20

6

The MIPS-lite Instruction Subset

- **ADDU and SUBU**

31	26	21	16	11	6	0
op	rs	rt	rd	shamt	funct	
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

 - addu rd,rs,rt
 - subu rd,rs,rt
- **OR Immediate:**

31	26	21	16	0
op	rs	rt	immediate	
6 bits	5 bits	5 bits	16 bits	

 - ori rt,rs,imm16
- **LOAD and STORE Word**

31	26	21	16	0
op	rs	rt	immediate	
6 bits	5 bits	5 bits	16 bits	

 - lw rt,rs,imm16
 - sw rt,rs,imm16
- **BRANCH:**

31	26	21	16	0
op	rs	rt	immediate	
6 bits	5 bits	5 bits	16 bits	

 - beq rs,rt,imm16

7/23/2012

Summer 2012 -- Lecture #20

7

Register Transfer Language (RTL)

- All start by *fetching* the instruction:
 R-format: {op, rs, rt, rd, shamt, funct} \leftarrow MEM[PC]
 I-format: {op, rs, rt, imm16} \leftarrow MEM[PC]
- RTL gives the meaning of the instructions:
Inst Register Transfers
 ADDU $R[rd] \leftarrow R[rs] + R[rt]; \text{ PC} \leftarrow \text{PC} + 4$
 SUBU $R[rd] \leftarrow R[rs] - R[rt]; \text{ PC} \leftarrow \text{PC} + 4$
 ORI $R[rt] \leftarrow R[rs] | \text{zero_ext}(\text{imm16}); \text{ PC} \leftarrow \text{PC} + 4$
 LOAD $R[rt] \leftarrow \text{MEM}[R[rs] + \text{sign_ext}(\text{imm16})]; \text{ PC} \leftarrow \text{PC} + 4$
 STORE $\text{MEM}[R[rs] + \text{sign_ext}(\text{imm16})] \leftarrow R[rt]; \text{ PC} \leftarrow \text{PC} + 4$
 BEQ if ($R[rs] == R[rt]$)
 then $\text{PC} \leftarrow \text{PC} + 4 + (\text{sign_ext}(\text{imm16}) || 00)$
 else $\text{PC} \leftarrow \text{PC} + 4$

7/23/2012

Summer 2012 -- Lecture #20

8

CPU开发过程概述

□ 5步骤

- ◆ 1) 分析每条指令的RTL, 梳理和总结出数据通路的设计需求
- ◆ 2) 选择恰当的数据通路功能部件
- ◆ 3) 组装数据通路根据指令RTL, 分析并建立功能部件间的正确连接关系
- ◆ 4) 根据指令RTL, 分析功能部件应执行的功能, 反推相应的控制信号取值
- ◆ 5) 生成控制器
 - 构造控制信号真值表, 然后推导出控制信号的最简表达式
 - 根据最简表达式构造门电路

计算机组成与实现

目录

- 概述
- 单周期CPU设计模型
- 数据通路基础部件建模
- 组装数据通路
- 控制介绍

计算机组成与实现

第1步：指令集功能需求

□ 存储器(MEM)

◆ 指令存储器 & 数据存储器(分离的存储器模拟了cache结构)

◆ 指令存储器只有读取数据（取指令）

◆ 数据存储器既有读取数据又有写入数据

□ 寄存器堆(32个32位寄存器)

◆ 能同时读出rs和rt两个寄存器

◆ 可以写数据至rt或rd寄存器

□ PC

□ 下一条指令地址的计算单元

□ 扩展立即数：符号/零扩展

□ 执行运算的计算单元

◆ Add/Sub/OR等

◆ 如何执行beq的比较操作？

中文	英文	缩写
数据存储器	Data Memory	DM
指令存储器	Instruction Memory	IM
寄存器堆	Register File	RF
程序计数器	Program Counter	PC
下指令地址	Next PC	NPC
扩展单元	Extender	EXT
算数逻辑单元	Arithmetic Logic Unit	ALU

计算机组成与实现

数据通路的抽象模型

□ 指令执行的主要步骤

◆ 取指令、译码/读操作数、执行、访存、回写

1.取指令 2.译码/读操作数 3.执行 4.访存 5.回写

注意

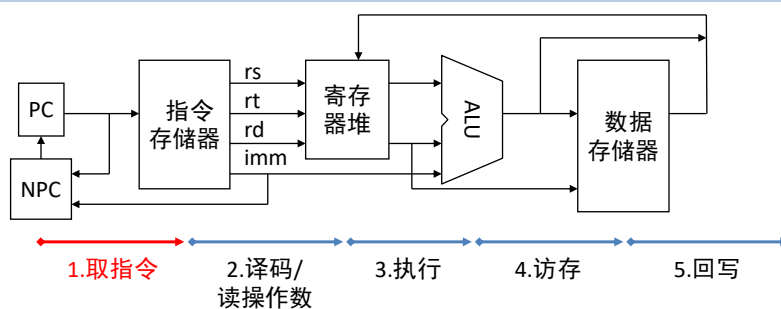
模型刻画了指令执行过程中的主要信息流的基本流动路径。

模型不是完整的设计。

对于分析更细微、更精确的设计细节的依赖与连接关系，模型具有重要指导意义。

计算机组成与实现

数据通路分解^{1/5}

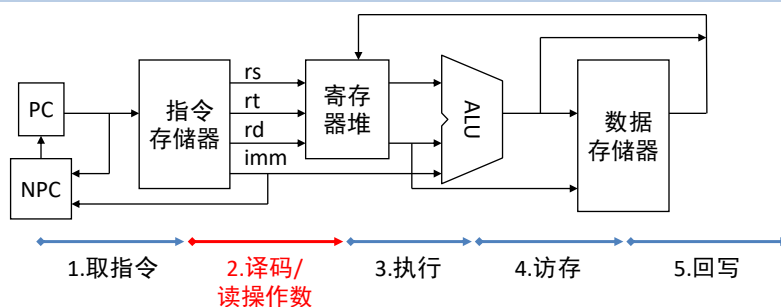


第1步：取指令（包含2个同时执行的功能）

- ◆ 功能1：PC驱动IM输出指令
- ◆ 功能2：PC驱动NPC计算下一个PC值

计算机组成与实现

数据通路分解^{2/5}

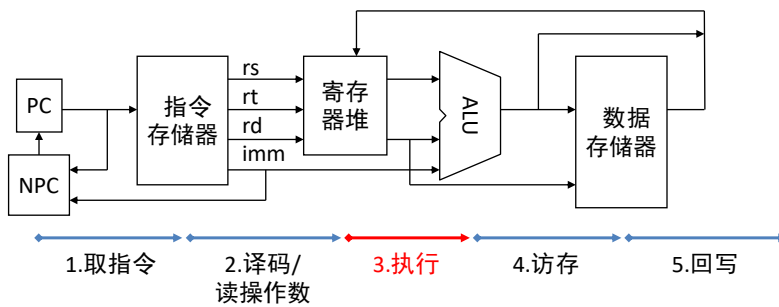


第2步：译码/读操作数（包含2个同时执行的功能）

- ◆ 功能1：IM驱动控制器（图中未画）分析指令的opcode和funct域
- ◆ 功能2：IM驱动RF读出2个寄存器值

计算机组成与实现

数据通路分解^{3/5}

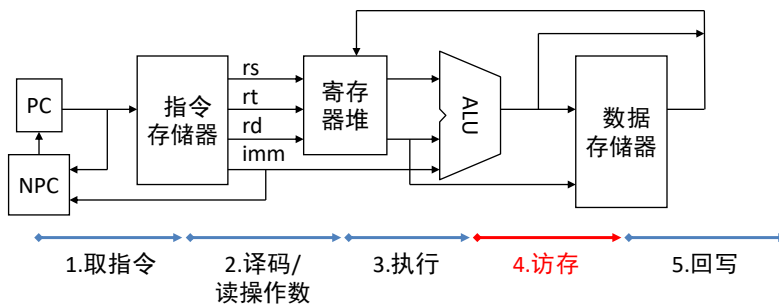


第3步：执行

- ◆ RF输出的寄存器值驱动ALU完成相应的计算
 - 算数运算(+, -, *, /), 移位, 逻辑(&, |), 比较(slt, ==)
 - 同时还承担计算lw和sw的地址

计算机组成与实现

数据通路分解^{4/5}

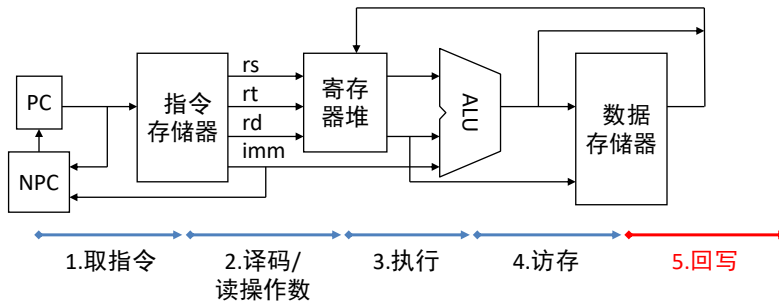


第4步：访存

- ◆ lw: DM输入地址后, 就输出数据
- ◆ sw: DM有2个输入, 地址&要写入的数据
- ◆ ※只有lw和sw指令在该环节有实际操作, 其他指令不涉及该环节

计算机组成与实现

数据通路分解^{5/5}

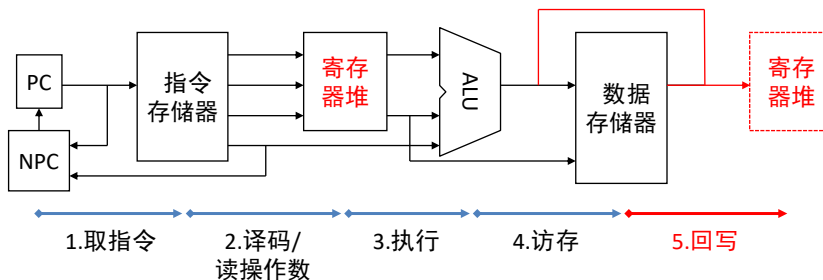


第5步：回写

- 操作：写ALU计算结果或数据存储器读出的数据至寄存器堆
- beq、jal、jr、sw不涉及该环节

计算机组成与实现

寄存器堆在数据通路中的独特地位



对于很多指令来说，它具有双重功效

- 它在第3阶段服务于读取操作数
- 它在第5阶段服务于回写结果

为逻辑清晰起见，可以“再部署”一个寄存器堆

- 前者代表读出，后者代表写入

计算机组成与实现

为什么是5个阶段？

- 是否可以有不同的阶段数？
 - ◆ 可以有不同的阶段数
 - 早期CPU的阶段数甚至只有2-3个
 - 现代CPU的阶段数甚至可能达到20-30个
- 为什么MIPS采用5阶段？
 - ◆ 虽然有些指令用不到5阶段，但lw却必须用到5阶段
 - 事实上，有些早期MIPS也不是5阶段

计算机组成与实现

目录

- 概述
- 单周期CPU设计模型
- 数据通路基础部件建模
 - ◆ MUX、寄存器
- 组装数据通路
- 控制介绍

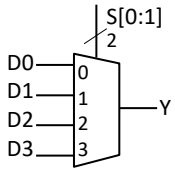
计算机组成与实现

MUX设计

1位2选1 MUX表达式: $Y = !S \& D0 + S \& D1$

1位4选1 MUX表达式: $Y = ?$
 $Y = !S0 \& !S1 \& D0 +$
 $!S0 \& S1 \& D1 +$
 $S0 \& !S1 \& D2 +$
 $S0 \& S1 \& D3$


1位N选1 MUX表达式?
 $Y = \sum_0^{N-1} S_i \& D_i$
Si: S的组合表示



Multiplexer (Mux)

- Selects between one of N inputs to connect to output
- $\log_2 N$ -bit select input – control input
- Example:

2:1 Mux



S	D ₁	D ₀	Y	S	Y
0	0	0	0	0	D ₀
0	0	1	1	1	D ₁
0	1	0	0		
0	1	1	1		
1	0	0	0		
1	0	1	0		
1	1	0	1		
1	1	1	1		

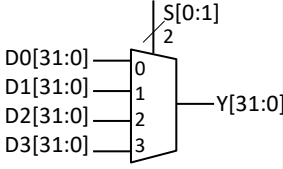
MUX设计

32位4选1 MUX表达式: ?

- 就是32个4选1 MUX
- Verilog表达式

```
assign Y = !S0 & !S1 & D0 +  
           !S0 & S1 & D1 +  
           S0 & !S1 & D2 +  
           S0 & S1 & D3 ;
```

M位N选1 MUX表达式?

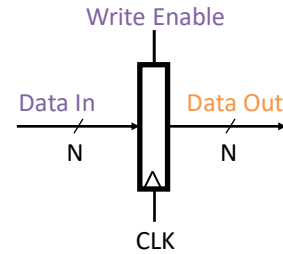


TIPS

表达式简写表达方法，即省略了[31:0]
前提：D0~D3及Y的位宽一致！

Storage Element: Register

- As seen in Logisim intro
 - N-bit input and output buses
 - Write Enable input
- Write Enable:
 - De-asserted (0): Data Out will not change
 - Asserted (1): Data In value placed onto Data Out on the rising edge of CLK



7/23/2012

Summer 2012 -- Lecture #20

23

VerilogHDL建模寄存器

□ 1位D寄存器：标准的寄存器

1位D寄存器

```

1 module d_ff( d, q, clk ) ;
2     input  d, clk ;
3     output q ;
4
5     reg    r ;
6
7     assign q = r ;
8     always @(posedge clk)
9         r <= d;
10
11 endmodule

```

r: 寄存器

将r从q输出
告诉编译器，以下行为按寄存器建模
时钟上升沿时，将输入保存至r

24

计算机组成与实现

VerilogHDL建模寄存器

1位写使能D寄存器

1位写使能D寄存器

```
1 module d_ff( d, we, q, clk ) ;
2     input  d, we ;
3     output q ;
4     input  clk ;
5
6     reg    r ;
7
8     assign q = r ;
9     always @( posedge clk )
10         if ( we )
11             r <= d;
12
13 endmodule
```

寄存器建模的完整写法:

```
if ( we )
    r <= d ;
else
    r <= r ;
```

Q: 为什么可以忽略else?

A: 对于缺少的分支, 编译器会
自动补充 “r <= r”。

25

计算机组成与实现

VerilogHDL建模寄存器

1位写使能D异步复位寄存器

1位写使能D异步复位寄存器

```
1 module d_ff( d, we, q, clk, rst ) ;
2     input  d, we ;
3     output q ;
4     input  clk, rst ;
5
6     reg    r ;
7
8     assign q = r ;
9     always @( posedge clk or posedge rst )
10         if ( rst )
11             r <= 1'b0 ;
12         else if ( we )
13             r <= d ;
14
15 endmodule
```

TIPS: 分析方法

由于rst在敏感表中, 因此rst的有效和clk的有效, 均会导致always语句块执行。这意味着rst对d的作用与clk无关, 故这就是异步复位。

Q: 在设计中如何选择异步复位or同步复位?

26

计算机组成与实现

VerilogHDL建模寄存器

32位写使能寄存器

32位写使能寄存器

```
1 module d32( d, we, q, clk ) ;
2     input  [31:0] d ;
3     input          we ;
4     output [31:0] q ;
5     input  clk ;
6
7     reg  [31:0] r ;
8
9     assign q = r ;
10    always @( posedge clk )
11        if ( we )
12            r <= d ;
13
14 endmodule
```

TIPS:
对于N位寄存器，仅仅改变输入信号、输出信号和内部寄存器定义的位数即可。

VerilogHDL建模寄存器

32位写使能寄存器（用generate-for建模。Verilog-2001标准）

32位写使能寄存器

```
1 module d32( d, we, q, clk ) ;
2     input  [31:0] d ;
3     input          we ;
4     output [31:0] q ;
5     input  clk ;
6
7     genvar          i ;
8
9     generate
10         for ( i=0; i<32; i=i+1 )
11             begin : label_d
12                 d_ff u_dff(d[i], we, q[i], clk) ;
13             end
14     endgenerate
15
16 endmodule
```

TIPS
1) genvar定义循环变量
2) 必须要有for、begin/end
3) begin必须要有标签

目录	
□	设计方法学概述
□	单周期CPU设计模型
□	数据通路基础部件建模
◆	PC
□	组装数据通路
□	控制介绍

计算机组成与实现

PC

□ PC非常简单，但也非常重要

□ PC本质上就是一个32位的寄存器

- ◆ 因为每条指令占用4B，所以PC的b1与b0恒为0
- ◆ 32位寄存器可以优化为30位寄存器

□ PC需要在系统复位后有一个确定的初值，即第1条指令的地址

- ◆ 这里先假设为0000_0000h

□ 根据上述分析，可以总结出PC的功能及输入输出信号

DI

32

DO

32

CLK

功能描述	Reset有效，寄存器置初值0x0000_0000。	
信号名	方向	描述
Clk	I	MIPS-C处理器时钟
Reset	I	复位信号
DI[31:0]	I	32位输入
DO[31:0]	O	32位输出

计算机组成与实现

目录

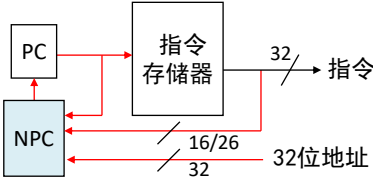
- 设计方法学概述
- 单周期CPU设计模型
- 数据通路基础部件建模
 - NPC
- 组装数据通路
- 控制介绍

计算机组成与实现

NPC（增加对jal和jr的支持）

- 任何指令的第一步除了取指令外，还要更新PC
- 更新PC，首先是NPC要计算出下一条指令的地址

指令	NPC执行的计算
顺序执行指令 addu/subu/ori/lw/sw	PC+4
分支或跳转指令 beq/jal/jr	beq 与PC和imm16相关 jal 与PC和imm26相关 jr 与rs的32位值相关



信号名	方向	描述
PC[31:0]	I	当前指令的地址
imm26[25:0]	I	jal: 26位偏移
A32[31:0]	I	jr: 保存的32位目标地址
NPC[31:0]	O	下一条指令的地址
NPCOp[1:0]	I	NPC计算模式的控制码

Q: NPC的输入信号是否充分?

计算机组成与实现

NPC（增加对jal和jr的支持）

任何指令的第一步除了取指令外，还要更新PC

更新PC，首先是NPC要计算出下一条指令的地址

指令	NPC执行的计算
顺序执行指令 addu/subu/ori/lw/sw	PC+4
分支或跳转指令 beq/jal/jr	beq 与PC和imm16相关
	jal 与PC和imm26相关
	jr 与rs的32位值相关

PC

NPC

指令存储器

32

32位地址

16/26

32

信号名	方向	描述
PC[31:0]	I	当前指令的地址
imm26[25:0]	I	jal: 26位偏移
A32[31:0]	I	jr: 保存的32位目标地址
NPC[31:0]	O	下一条指令的地址
NPCOp[1:0]	I	NPC计算模式的控制码
Zero	I	rs与rt相等的比较结果

beq的RTL

PC ←

(rs==rt) ? PC+4+sext(imm16) :

PC+4

(rs==rt): PC计算涉及一个比较结果

计算机组成与实现

目录

设计方法学概述

单周期CPU设计模型

数据通路基础部件建模

寄存器堆

组装数据通路

控制介绍

计算机组成与实现

17

寄存器堆的设计考虑

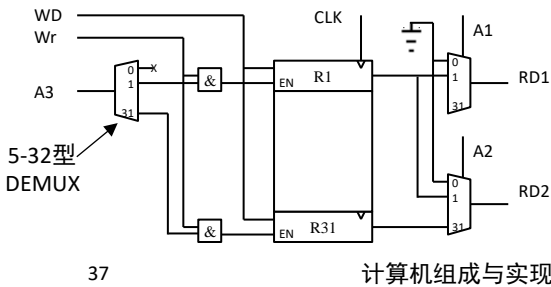
□ 示例：2-4型DEMUX

◆ A为输入2位，Y3至Y0均为1位输出

A	Y3	Y2	Y1	Y0
00	0	0	0	1
01	0	0	1	0
10	0	1	0	0
11	1	0	0	0



$$\begin{aligned} Y3 &= \overline{A1} \cdot \overline{A0} \\ Y2 &= \overline{A1} \cdot A0 \\ Y1 &= A1 \cdot \overline{A0} \\ Y0 &= A1 \cdot A0 \end{aligned}$$



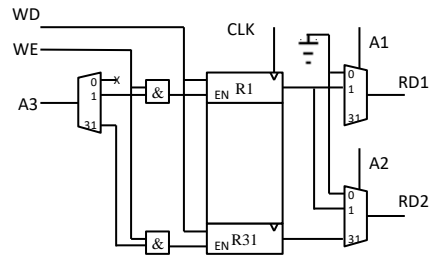
37

计算机组成与实现

Verilog建模RF

□ 用行为建模方法建模RF

□ RF写入语句利用了RW是输入信号（即RW是变值）这一特性



```
reg [31:0] rf[31:1] ;

always @( posedge clk )
    if ( WE )
        rf[A3] <= WD ;

assign RD1 = (A1==0) ? 32'b0 :
                rf[A1] ;
assign RD2 = ...
```

38

计算机组成与实现

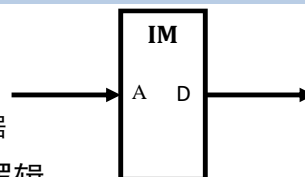
目录

- 设计方法学概述
- 单周期CPU设计模型
- 数据通路基础部件建模
 - ◆ 指令存储器
- 组装数据通路
- 控制介绍

计算机组成与实现

指令存储器

- 存储器可以理解为一个数组
 - ◆ 存储器的地址就是数组的下标
 - ◆ 给出存储器地址，存储器就输出对应单元的数据
- 执行读出操作时，存储器行为可视为组合逻辑
 - ◆ 地址A有效一段时间后，数据RD就输出正确的值
 - 这个所谓的“一段时间”被称为访问时间



有关不同存储器类型的详细内容，在存储层次中介绍

access time~访问时间

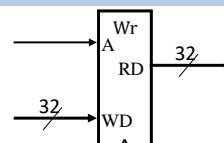
计算机组成与实现

目录

- 设计方法学概述
- 单周期CPU设计模型
- 数据通路基础部件建模
 - ◆ 数据存储器
- 组装数据通路
- 控制介绍

数据存储器

- 与指令存储器不同，数据存储器要支持写入
 - ◆ WD：写入的数据
- 写使能
 - ◆ 与寄存器堆类似，数据存储器需要有写使能信号Wr
- 存储器访问
 - ◆ 读：Wr=0，A单元数据从RD输出
 - 读出操作时可视为组合逻辑，即A有效一段时间后，RD就输出正确的值
 - ◆ 写：在时钟上升沿时，如果Wr=1，则WD被写入A单元中
- *CLK只对写操作有效，对读操作无效



Verilog建模存储器

- 建模要点：内部是reg阵列
- 时序特点：写入的数据滞后1个cycle输出
 - ◆ 由寄存器特性决定

```

1 module MEM4KB( A, DI, We, DO, clk ) ;
2     input  [9:0] A;
3     input  [31:0] DI ;
4     input    We ;
5     output [31:0] DO ;
6     input    clk ;
7
8     reg     [31:0] array[1023:0] ;
9
10    assign DO = array[A] ;
11
12    always @( posedge clk )
13        if ( We )
14            array[A] <= DI ;
15 endmodule

```

TIPS:

建模类似于RF。
实际设计芯片时，
会采用定制的库，
而不会用寄存器
方式实现存储器。

对于P8，存储器
要用FPGA芯片内
置的块存储器。

43



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

目录

- 设计方法学概述
- 单周期CPU设计模型
- 数据通路基础部件建模
 - ◆ ALU
- 组装数据通路
- 控制介绍

计算机组成与实现

ALU需求分析

计算需求：加、减、或、相等

相等

方法1：设计独立的比较电路，例如利用XOR运算

方法2：执行减法运算，然后再判断结果是否全0

采用方法2，减法电路被重用

指令	RTL描述
addu	$R[rd] \leftarrow R[rs] + R[rt]; PC \leftarrow PC + 4$
subu	$R[rd] \leftarrow R[rs] - R[rt]; PC \leftarrow PC + 4$
ori	$R[rt] \leftarrow R[rs] \mid \text{zero_ext}(\text{imm16}); PC \leftarrow PC + 4$
lw	$R[rt] \leftarrow \text{MEM}[R[rs] + \text{sign_ext}(\text{imm16})]; PC \leftarrow PC + 4$
sw	$\text{MEM}[R[rs] + \text{sign_ext}(\text{imm16})] \leftarrow R[rt]; PC \leftarrow PC + 4$
beq	$\text{if } (R[rs] == R[rt])$ $\text{then } PC \leftarrow PC + 4 + (\text{sign_ext}(\text{imm16}) \mid 00)$ $\text{else } PC \leftarrow PC + 4$

计算机组成与实现

减法运算

计算 $Y=A-B$ ，可以将其等价转换为 $Y=A+(-B)$

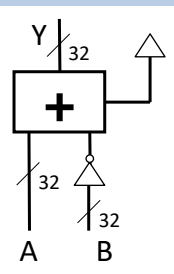
利用二进制补码的负数转换规则：
 $-B = \bar{B} + 1$

因此
 $Y = A - B = A + \bar{B} + 1$

可以利用加法器构造减法运算

B输入取反

Cin为1



46

计算机组成与实现

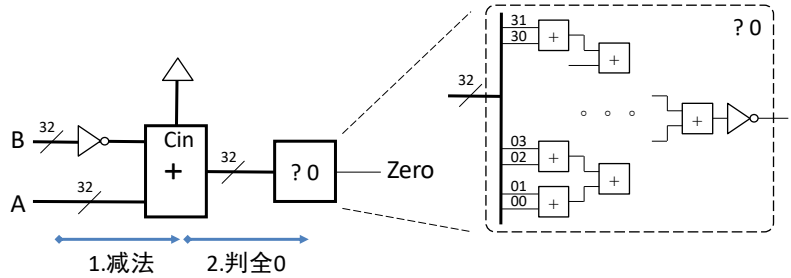
23

相等

- 采用重用减法的思路
 - 先执行减法，然后再判断结果是否全0
- Zero
 - 0: $A \neq B$
 - 1: $A = B$

问题1
判0电路需要多少OR门？

问题2
如果采用XOR实现相等，需要多少AND门、OR门、NOT门？



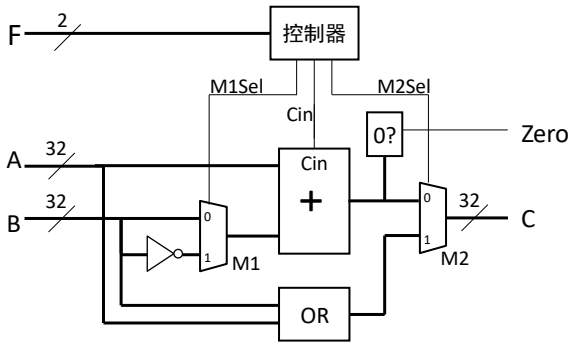
47

计算机组成与实现

集成

- M1: 0通道对应加法，1通道对应减法
 - 加法: $A + B$
 - 减法: $A - B = A + \bar{B} + 1$
- M2: 0通道对应加减法的结果，1通道对应OR的结果
- Cin: 如果执行加法则为0，如果执行减法则为1

F	功能
00	加
01	减
10	或



根据控制器的功能表，结合ALU内部结构设计，可以建立内部各控制信号的真正表，进而推导出表达式，从而构造出控制器的门电路结构

48

计算机组成与实现

VerilogHDL建模4位加法与减法

4位加法（无overflow检测）

4位加法

```
1 module add4( a, b, c ) ;
2     input  [3:0]  a, b ;
3     output [3:0]  c ;
4
5     assign c = a + b ;
6
7 endmodule
```

TIPS:

Verilog的“+”：最基础的二进制加法，是不区分正负数的！

4位减法（采用二进制补码运算方法，即加相反数）

4位减法（无overflow检测）

```
1 module sub4( a, b, c ) ;
2     input  [3:0]  a, b ;
3     output [3:0]  c ;
4
5     assign c = a + ~b + 1'b1 ;
6
7 endmodule
```

TIPS:

也可以采用
`assign c = a - b ;`



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

VerilogHDL建模4位ALU

ALU的功能：加、减、或

- ◆ 加/减：不支持overflow

head.v

```
`define ALU_ADDU 2'b00
`define ALU_SUBU 2'b01
`define ALU_OR   2'b10
```

op：控制信号

- ◆ 00~加法；01~减法；10~或；11~保留

建模方法：利用assign语句实现加法与减法的2选1

```
1 `include "head.v"
2
3 module ALU( a, b, c, op ) ;
4     input  [3:0]  a, b ;
5     input  [1:0]  op ;
6     output [3:0]  c ;
7
8     assign c = (op==`ALU_ADDU) ? (a + b)      : 加
9                  (op==`ALU_SUBU) ? (a + ~b + 1) : 减
10                  (op==`ALU_OR)  ? (a | b)      : 或
11                                     4'b0000 ; 保留
12 endmodule
```

计算机学院



目录

- 概述
- 单周期CPU设计模型
- 数据通路基础部件建模
- 组装数据通路
- 控制介绍

计算机组成与实现

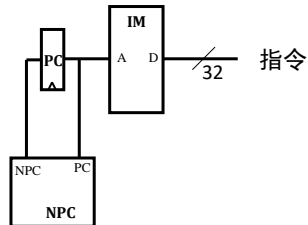
第3步：组装数据通路

- 每条指令都有各自的功能需求，对应的功能部件及其连接关系也是不同的
- 数据通路就是所有功能部件及其连接关系的集合
- 如何根据RTL需求组装数据通路？
 - ◆ 根据指令的RTL，选取和添加相应的部件，并建立正确的连接关系
 - ◆ 重复上述过程

计算机组成与实现

数据通路的共性阶段

- 所有指令都需要：取指令（IF）
 - ◆ PC驱动IM，IM输出指令
 - ◆ PC驱动NPC，NPC计算PC+4然后再输出至PC（目的是更新PC）



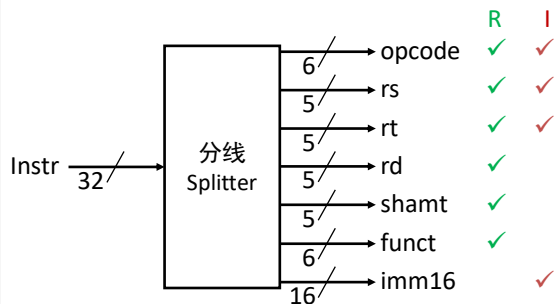
※ 现阶段先假设所有指令都是顺序执行的

计算机组成与实现

32位指令的分解

- 将指令的32位信号分解为各个域
- 原理：类似于一路入户电分成多路室内电
- 以Instr[5:0]为例
 - ◆ 分一路为imm的最低5位
 - ◆ 另一路为funct

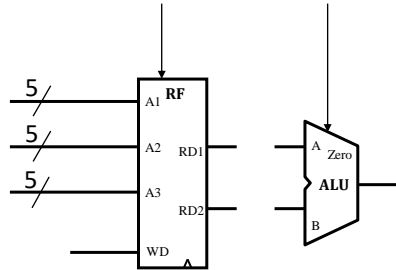
```
assign funct = Instr[05:00]
assign imm16 = Instr[15:00]
assign imm26 = Instr[25:00]
```



Q:
1位信号，分多少路都容易。
但多位信号分若干路，如何
确保信号间不会产生交叉
（短路）？

第3步：ADDU & SUB

- **ADDU:** $R[rd] \leftarrow R[rs] + R[rt]$
- 硬件需求：
 - ◆ 寄存器堆：2路读出信号，1路写回信号
 - ◆ ALU：执行加/减



计算机组成与实现

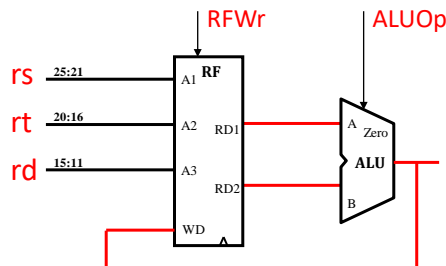
第3步：ADDU & SUB

- **ADDU:** $R[rd] \leftarrow R[rs] + R[rt]$

Q1: RFWr取值为0还是1?
Q2: ALUOp取值为为什么?

- 连接

- ◆ 寄存器堆的输出 \leftrightarrow ALU的输入
- ◆ 指令分解出的rs/rs/rd \leftrightarrow 寄存器堆的A1/A2/A3
- ◆ 寄存器堆的写使能 \leftrightarrow RFWr; ALU的控制码 \leftrightarrow ALUOp



计算机组成与实现

第3步：ORI

ORI: $R[rt] \leftarrow R[rs] \text{ OR } \text{zero_ext}(\text{imm16})$

硬件需求

- zero_ext(): 这是一个新的计算需求
 - 原有的功能部件无法满足该需求

计算机组成与实现

第3步：ORI

ORI: $R[rt] \leftarrow R[rs] \text{ OR } \text{zero_ext}(\text{imm16})$

硬件需求

- zero_ext(): 这是一个新的计算需求
 - 原有的功能部件无法满足该需求
- EXT: 新增功能部件，用于将16位数进行0扩展为32位数

信号名称	方向	描述
Imm16[15:0]	输入	16位输入。
Ext[31:0]	输出	32位0扩展结果。

HDL建模：Extender.v

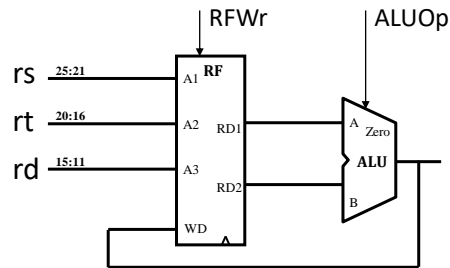
```
module EXT( Imm16, Ext ) ;  
    . . .  
    assign Ext = {16{0}, Imm16} ;  
end module
```

第3步：ORI

❑ ORI: $R[rt] \leftarrow R[rs] \text{ OR } \text{zero_ext}(\text{imm16})$

❑ 硬件需求

- ◆ 如何传递32位扩展结果给ALU?
- ◆ 如何让ALU结果写入rt而不是rd?



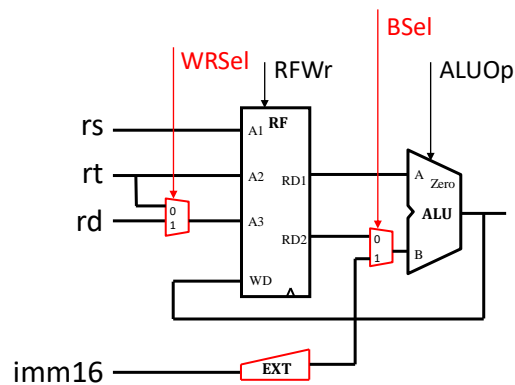
计算机组成与实现

第3步：ORI

❑ ORI: $R[rt] \leftarrow R[rs] \text{ OR } \text{zero_ext}(\text{imm16})$

❑ 增加新硬件

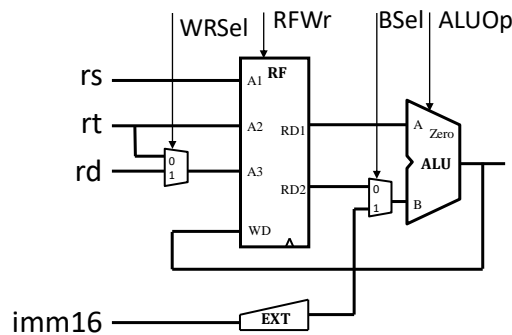
- ◆ EXT、2个MUX



计算机组成与实现

第3步：LW

- **LW:** $R[rt] \leftarrow MEM[R[rs] + \text{sign_ext}(imm16)]$
- 硬件需求
 - ◆ `sign_ext()`: 这是一个新的计算需求
 - EXT无法满足该需求



计算机组成与实现

第3步：LW

- **LW:** $R[rt] \leftarrow MEM[R[rs] + \text{sign_ext}(imm16)]$
- 硬件需求
 - ◆ `sign_ext()`: 这是一个新的计算需求
 - ◆ `zero_ext()`与`sign_ext()`, 其输入位数、输出位数及基本目的均相同
 - ◆ 根据“高内聚、低耦合”原则, 由EXT同时实现两种扩展较为合理
 - 由于EXT同时支持两种扩展, 因此必须增加控制信号`EXTOp`

信号名称	方向	描述
Imm[15:0]	输入	16位输入。
EXTOp	输入	扩展功能选择 0: 符号扩展 1: 无符号扩展
Ext[31:0]	输出	32位0扩展结果。

计算机组成与实现

第3步：LW

- **LW:** $R[rt] \leftarrow MEM[R[rs] + sign_ext(imm16)]$
- 硬件需求
 - ◆ `sign_ext()`: 这是一个新的计算需求
 - ◆ `zero_ext()`与`sign_ext()`, 其输入位数、输出位数及基本目的均相同
 - ◆ 根据“高内聚、低耦合”原则, 由EXT同时实现两种扩展较为合理
 - 由于EXT同时支持两种扩展, 因此必须增加控制信号`EXTOp`

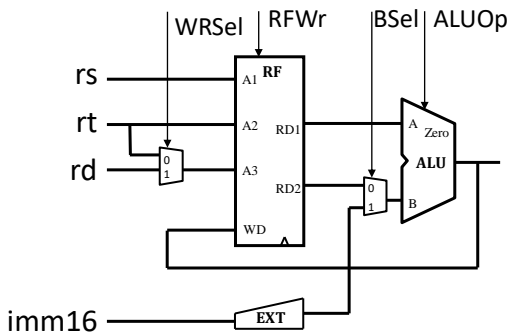
HDL建模: Extender.v

```
module EXT( Imm, F, Ext ) ;  
    ...  
    assign Ext = EXTOp==`ZEXT ? {16{0}, Imm} :  
                                   {16{Imm[15]}, Imm} ;  
end module
```

书写表达式时, 尽量多
使用宏, 增加可读性。

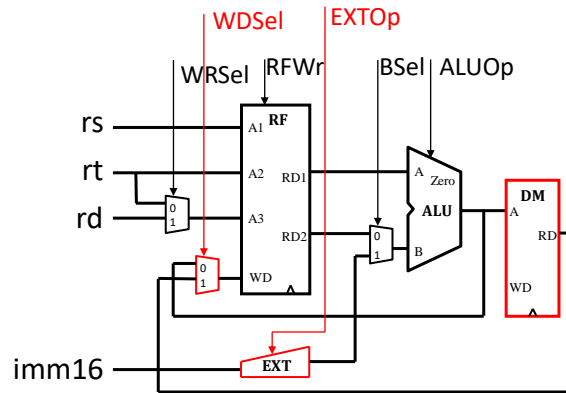
第3步：LW

- **LW:** $R[rt] \leftarrow \text{MEM}[R[rs] + \text{sign_ext}(imm16)]$
- 硬件需求
 - ◆ 需要DM



第3步: LW

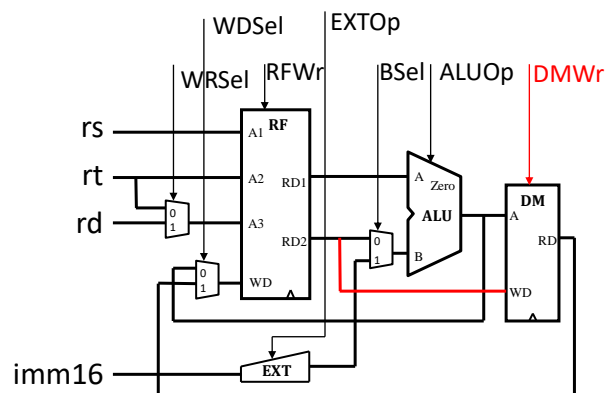
- ❑ **LW:** $R[rt] \leftarrow \text{MEM}[R[rs] + \text{sign_ext}(\text{imm16})]$
- ❑ 硬件需求
 - ◆ 需要DM
 - ◆ 需要回写RF



计算机组成与实现

第3步: SW

- SW: $\text{MEM}[\text{R}[\text{rs}] + \text{sign_ext}(\text{imm16})] \leftarrow \text{R}[\text{rt}]$
- 连接
 - RF的第2个输出 \leftrightarrow DM的WD



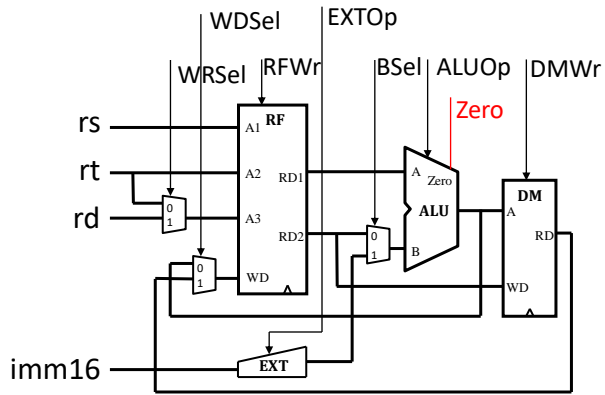
计算机组成与实现

第3步：BEQ

□ 本质上，beq涉及2大功能

- ◆ 功能1：寄存器比较
 - 让ALU执行减法，然后把比较结果zero传递给NPC

```
if ( R[rs] == R[rt] )  
    PC ← PC+4 + sign_ext(imm16||00)  
else  
    PC ← PC+4
```



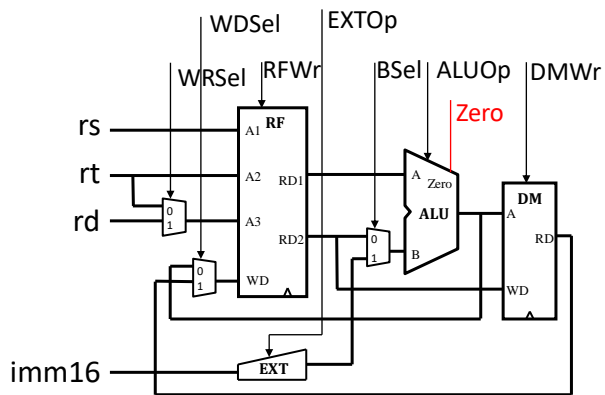
计算机组成与实现

第3步：BEQ

□ 本质上，beq涉及2大功能

- ◆ 功能2：根据比较的结果计算PC
 - 这属于NPC的功能范畴

```
if ( R[rs] == R[rt] )  
    PC ← PC+4 + sign_ext(imm16||00)  
else  
    PC ← PC+4
```



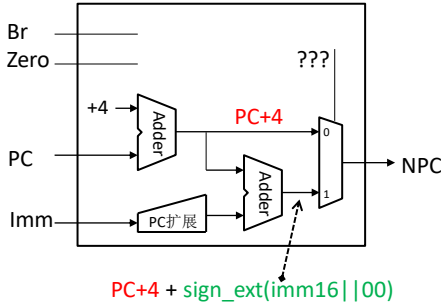
计算机组成与实现

第3步：BEQ

□ 本质上，beq涉及2大功能

- ◆ 功能2：根据比较的结果，计算PC。这属于NPC的功能范畴

□ 对于NPC，现在需要知道当前指令是否是beq及zero的结果



注意：
此处NPC设计，未考虑jal和jr指令！

```
if ( R[rs] == R[rt] )
    PC ← PC+4 + sign_ext(imm16||00)
else
    PC ← PC+4
```

信号名	方向	描述
PC[31:0]	I	32位输入
Imm[15:0]	I	16位立即数
Br	I	beq指令标志 1: 当前指令是beq 0: 当前指令不是beq
Zero	1	rs和rt相等标志 1: 相等 0: 不等
NPC[31:0]	O	32位输出

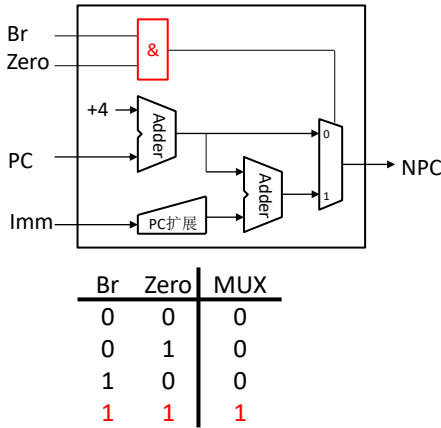
计算机组成与实现

第3步：BEQ

□ 本质上，beq涉及2大功能

- ◆ 功能2：根据比较的结果，计算PC。这属于NPC的功能范畴

□ 对于NPC，现在需要知道当前指令是否是beq及zero的结果



```
if ( R[rs] == R[rt] )
    PC ← PC+4 + sign_ext(imm16||00)
else
    PC ← PC+4
```

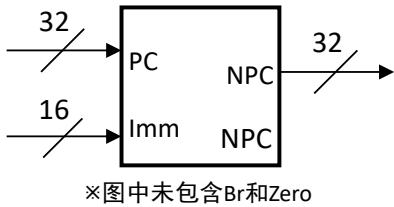
信号名	方向	描述
PC[31:0]	I	32位输入
Imm[15:0]	I	16位立即数
Br	I	beq指令标志 1: 当前指令是beq 0: 当前指令不是beq
Zero	1	rs和rt相等标志 1: 相等 0: 不等
NPC[31:0]	O	32位输出

计算机组成与实现

第3步：BEQ

- 本质上，beq涉及2大功能
 - 功能2：根据比较的结果，计算PC。这属于NPC的功能范畴
- 对于NPC，现在需要知道当前指令是否是beq及zero的结果

```
if ( R[rs] == R[rt] )
    PC←PC+4 + sign_ext(imm16||00)
else
    PC←PC+4
```



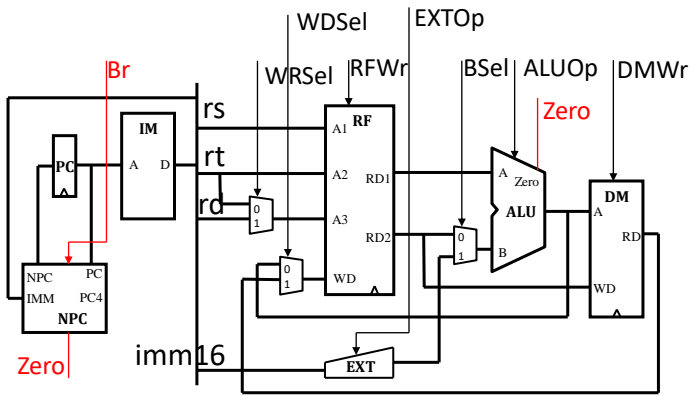
信号名	方向	描述
PC[31:0]	I	32位输入
Imm[15:0]	I	16位立即数
Br	I	beq指令标志 1: 当前指令是beq 0: 当前指令不是beq
Zero	1	rs和rt相等标志 1: 相等 0: 不等
NPC[31:0]	O	32位输出

计算机组成与实现

第3步：BEQ

- 组装

```
if ( R[rs] == R[rt] )
    PC←PC+4 + sign_ext(imm16||00)
else
    PC←PC+4
```



计算机组成与实现

目录

- 概述
- 单周期CPU设计模型
- 数据通路基础部件建模
- 组装数据通路
- 控制介绍

计算机组成与实现

CPU开发过程概述

- 5步骤
 - ◆ 1) 分析每条指令的RTL，梳理和总结出数据通路的设计需求
 - ◆ 2) 选择恰当的数据通路功能部件
 - ◆ 3) 组装数据通路根据指令RTL，分析并建立功能部件间的正确连接关系
 - ◆ 4) 根据指令RTL，分析功能部件应执行的功能，反推相应的控制信号取值
 - ◆ 5) 生成控制器
 - 构造控制信号真值表，然后推导出控制信号的最简表达式
 - 根据最简表达式构造门电路

计算机组成与实现

Control

- Need to make sure that correct parts of the datapath are being used for each instruction
 - Have seen *control signals* in datapath used to select inputs and operations
 - For now, focus on what value each control signal should be for each instruction in the ISA
 - Next lecture, we will see how to implement the proper combinational logic to implement the control

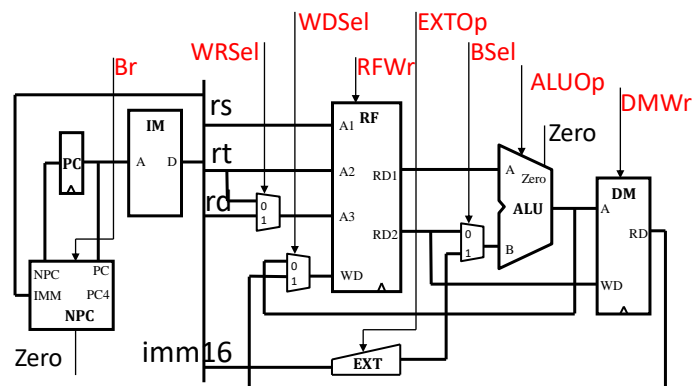
7/23/2012

Summer 2012 -- Lecture #20

75

数据通路的控制信号

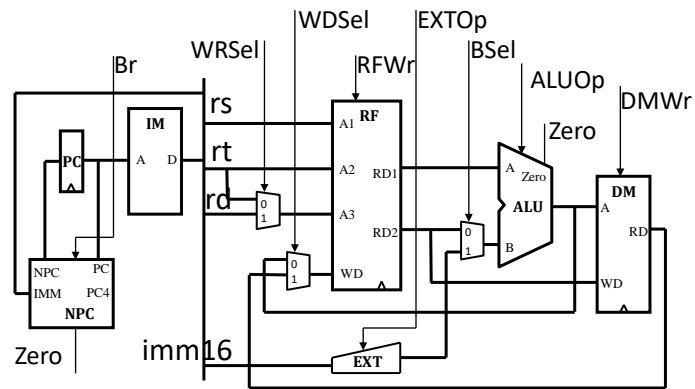
EXTOp:	0 → "zero"; 1 → "sign"	DMWr:	1 → write memory
BSel:	0 → busB; 1 → imm16	WDSel:	0 → ALU; 1 → Mem
ALUOp:	"ADD", "SUB", "OR"	WRSel:	0 → "rt"; 1 → "rd"
Br:	0 → +4; 1 → branch	RFWr:	1 → write register



计算机组成与实现

执行路径: addu

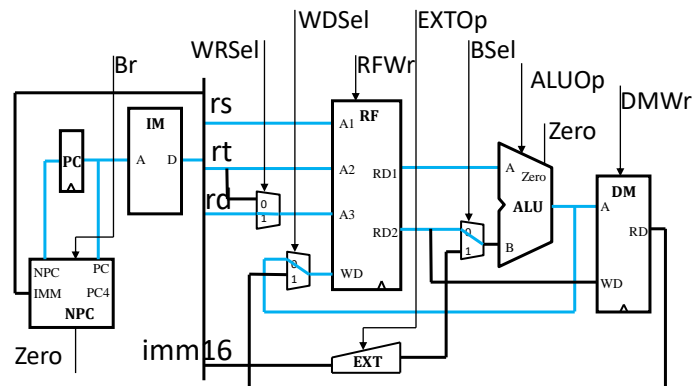
ADDU: $R[rd] \leftarrow R[rs] + R[rt]$



计算机组成与实现

执行路径: addu

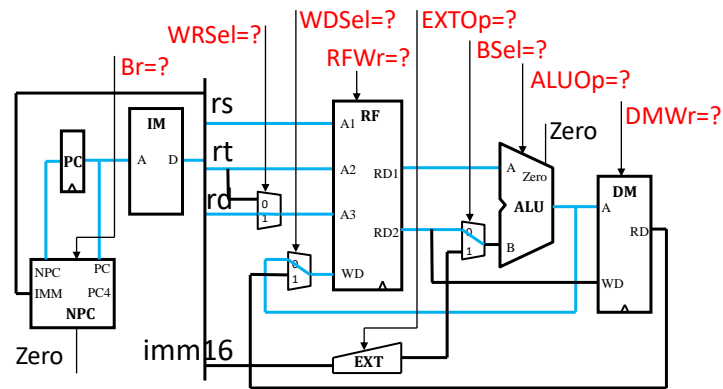
ADDU: $R[rd] \leftarrow R[rs] + R[rt]$



计算机组成与实现

执行路径: addu

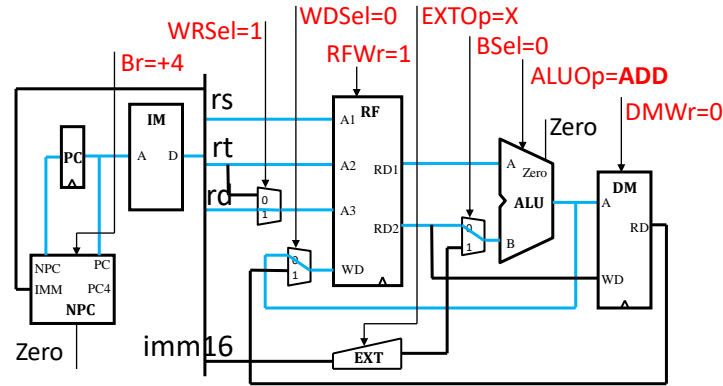
ADDU: $R[rd] \leftarrow R[rs] + R[rt]$



计算机组成与实现

执行路径: addu

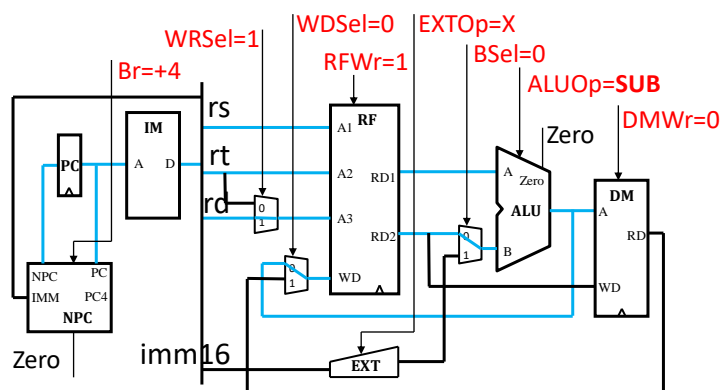
ADDU: $R[rd] \leftarrow R[rs] + R[rt]$



计算机组成与实现

执行路径：subu

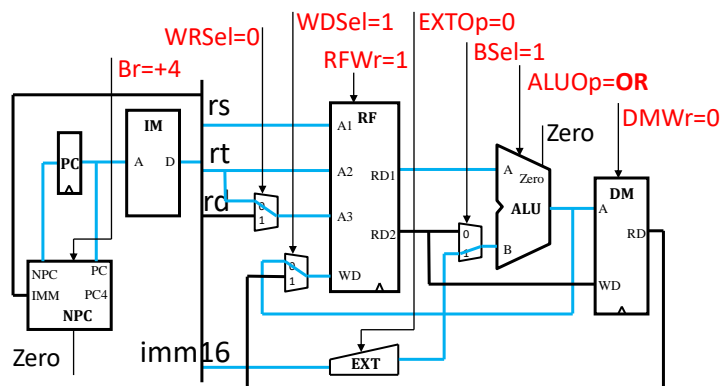
□ SUBU: $R[rd] \leftarrow R[rs] - R[rt]$



计算机组成与实现

执行路径：ori

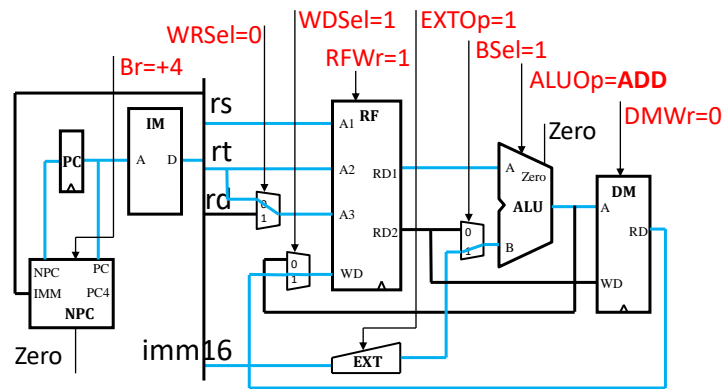
□ ORI: $R[rt] \leftarrow R[rs] \text{ OR } \text{zero_ext}(\text{imm16})$



计算机组成与实现

执行路径：lw

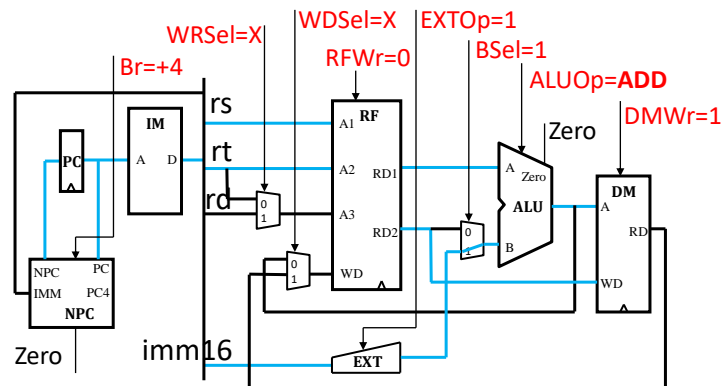
□ **LW:** $R[rt] \leftarrow \text{MEM}[R[rs] + \text{sign_ext}(\text{imm16})]$



计算机组成与实现

执行路径：sw

□ **SW:** $\text{MEM}[R[rs] + \text{sign_ext}(\text{imm16})] \leftarrow R[rt]$

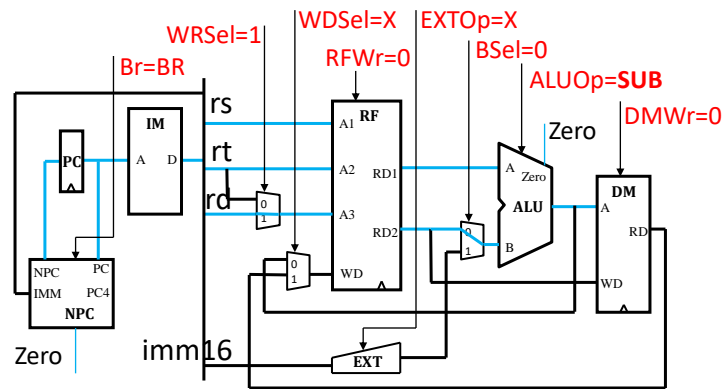


计算机组成与实现

执行路径: beq

BEQ

```
if ( R[rs] == R[rt] )
    PC ← PC+4 + sign_ext(imm16||00)
else
    PC ← PC+4
```

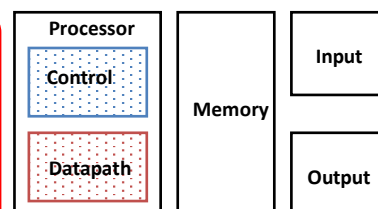


计算机组成与实现

Summary (1/2)

Five steps to design a processor:

- 1) Analyze instruction set → datapath requirements
- 2) Select set of datapath components & establish clock methodology
- 3) Assemble datapath meeting the requirements
- 4) Analyze implementation of each instruction to determine setting of control points that effects the register transfer
- 5) Assemble the control logic
 - Formulate Logic Equations
 - Design Circuits



Summary (2/2)

- Determining control signals
 - Any time a datapath element has an input that changes behavior, it requires a control signal (e.g. ALU operation, read/write)
 - Any time you need to pass a different input based on the instruction, add a MUX with a control signal as the selector (e.g. next PC, ALU input, register to write to)
- Your datapath and control signals will change based on your ISA